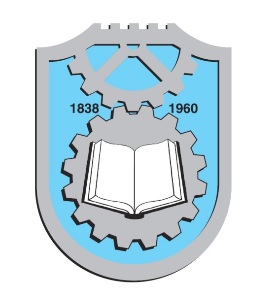
Универзитет у Крагујевцу

Факултет инжењерских наука



Предмет

Вештачка интелигенција

Пројектни задатак

**Предвиђање смера возова коришћењем неуронских мрежа**

Предавачи Студент

др Весна Ранковић, ред. Проф. Милић Ленка 579/2015

Тијана Шуштершич, стручни сарадник

Садржај

1. Опис проблема.............................................................................................................3

2. Потребни софтвери и библиотеке………………………………………………………………………….4

3. Обрада и припрема података.....................................................................................8

4. Архитектура неуронске мреже.................................................................................11

4. 1. Активационе функције…………………………………………………………………………….13

4. 2. Параметри за тренирање………………………………………………………………………..16

5. Неуронска мрежа са L слојева………..........................................................................17

5. 1. Закључак…….................................................................................................24

6. Оптимизована неуронска мрежа са L слојева……….................................................24

6. 1. Иницијализација параметара / Дефинисање параметара………………….24

6. 2. Иницијализација оптимизатора…………………………………………………………..29

6. 3. Петља кроз број епоха………………………………………………………………………….25

6. 3. 1. Избор мини серија улазних и излазних података ……………..……25

6. 3. 2. Петља за сваки пар серија података………………………………………..25

6. 3. 3. Пропагација унапред………………………………………………………………..32

6. 3. 4. Рачунање трошка параметара………………………………………………….35

6. 3. 5. Пропагација уназад…………………………………………………………………..36

6. 3. 6 Избор оптимизационог алгоритма……………………………………………41

6. 3. 6. 1. Update-овање параметара………………………………………….41

7. Модел неуронске мреже са L слојева………………………………………………………………..…45

8. Тренирање и резултати неуронске мреже..............................................................47

9. Закључак.....................................................................................................................51

10. Литература................................................................................................................52

1. Опис проблема

Задатак овог рада састоји се у предвиђању да ли ће воз бити на истоку или на западу. Воз садржи променљив број вагона (*instances,* слика 1) који су различитог облика и носе различита оптерећења. Наведене параметре описују атрибути (*features,* слика 1).

Атрибути који описују вагоне су :

*C0, C1, C2, C3, C4, C5, L0, L1* и *Train\_list1\_order* (слика 2).

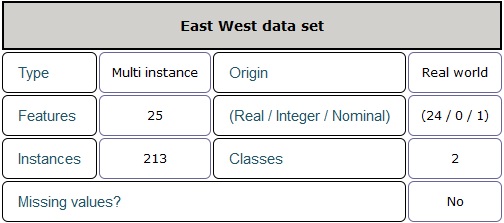
Описе свих атрибута можете пронаћи у приложеном фајлу *“East West dataset.txt”* или на следећем линку:

<https://sci2s.ugr.es/keel/dataset.php?cod=167&fbclid=IwAR3GAFEghWKVo_U5bF787zrlR8EnAxGGhfCwOdbE4erdCYfiYfJFoFSMwCk#sub1>. Они се тичу броја вагона, описа сваког од вагона тј. каквог су облика вагони кад се гледају одозго, облика пртљага итд.

Атрибути представљају улазне параметре у неуронску мрежу. Укупан број атрибута је 25 за сваки од улаза мреже (за сваку од инстанци) (слика 1).

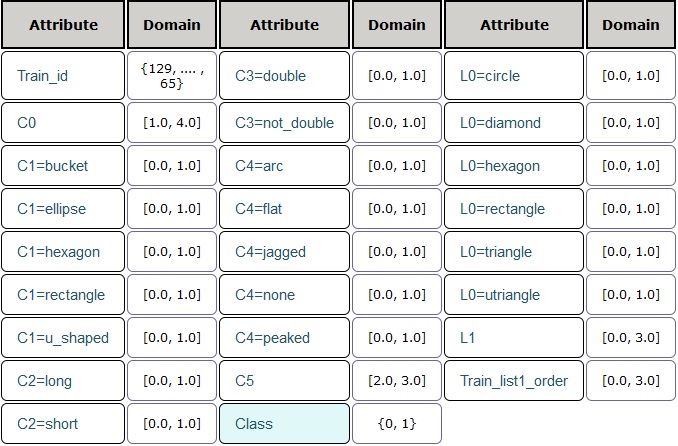
Број инстанци, који дефинишу укупан број примера у проблему *исток/запад*, је 213.

Излаз неуронске мреже је бинарна вредност (0 или 1), где излаз 1 дефинише воз који иде на исток, а 0 воз који иде на запад. На слици 1опис излаза мреже представљен је класом која има две вредности (0 и 1).



*Слика 1 :*

*Опис скупа података*



*Слика 2 :*

*Вредности улазних података и вредности излаза неуронске мреже*

2. Потребни софтвери и библиотеке

- Anaconda (платформа за Python науку о подацима)

- numpy

- scikit-learn

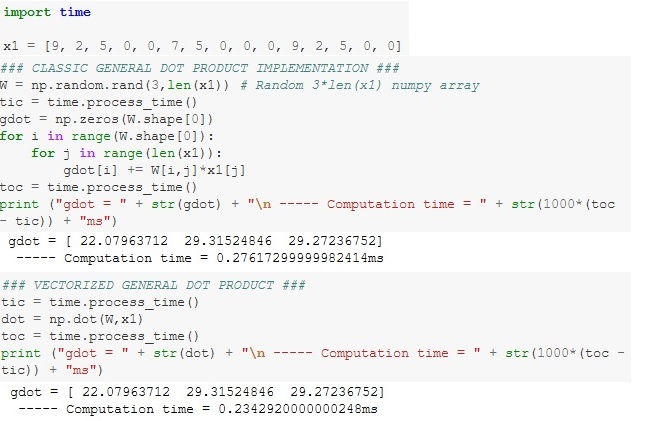
- matplotlib

- jupyter notebook

Јако је важно да се при великом броју података код ефективно и брзо извршава јер је у области дубоког учења главна карактеристика велики број тренинг примера у којима се техника дубоког учења добро показала као погодна метода за решавање проблема са великим, улазним скупом података. Због овог разлога за проблем *исток/запад* коришћена је python библиотека numpy која служи за брзе компутације између n-димензионих вектора. Компутације се, на нивоу вектора, имплементирају помоћу векторизације и broadcast-инга коју омогућава ова библиотека.

Уметност векторизације у python-у омогућава процесуирање n-димензионих вектора без коришћења foor петљи.

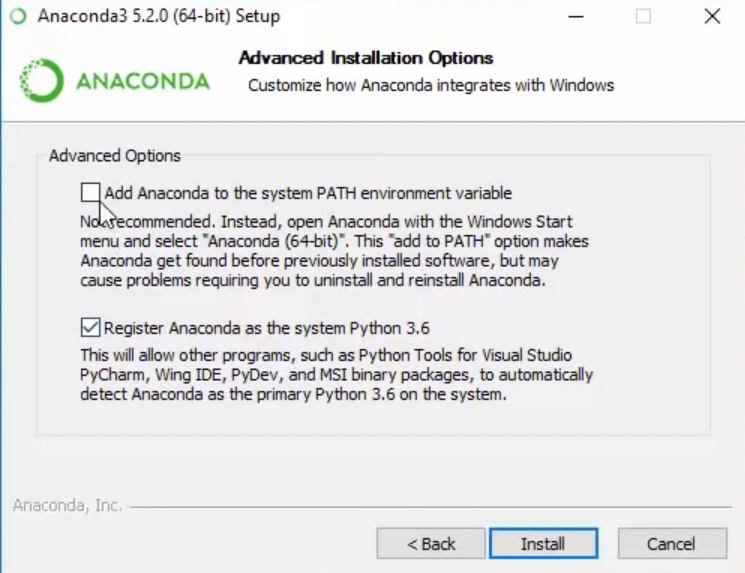
Множење вектора (операција која је већ уграђена у numpy векторизованим функцијама), као и многе друге математичке операције, омогућавају до 10 пута брже извршавање од оног када би се користила foor петља зарад истог резултата. (слика 3).



*Слика 3:*

*Време извршавања foor петље у поређењу са векторизованом numpy операцијом*

Дакле, далеко је боље користити библиотеку numpy како би код добио на брзини извршавања и како би избегли спору foor петљу која је у области дубоког учења јако непожељна.

Проблем *исток/запад* имплементиран је коришћењем Anaconda 2019.07 платформе која представља један од најлакших начина за извођење Python науке о подацима. Користи се за развијање, тестирање и тренинг на рачунарима и омогућава брзо преузимање Python библиотека, анализу података, визуализацију резултата итд. На следећем линку могуће је преузети једну од Anaconda дистрибуција:

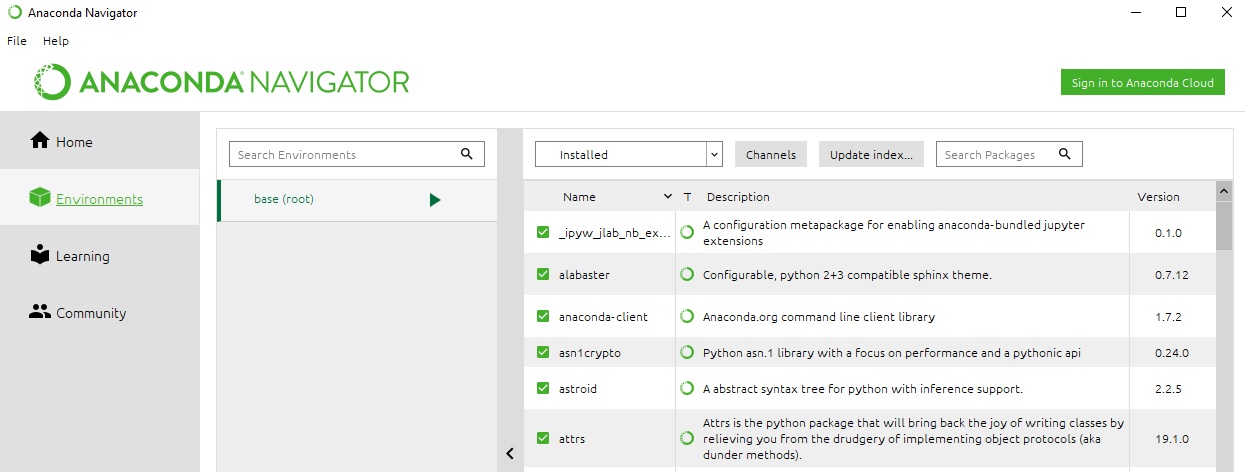
<https://www.anaconda.com/distribution/>

У току инсталације добро је напоменути да се квадрат поред описа *“Add Anaconda to the system PATH environment variable”* не селектује уколико на рачунару већ постоји инсталиран Python (слика 4).

*Слика 4:*

*Инсталација Anaconda платформе*

Након инсталације у Windows старт менију укуцајте Anaconda и избором Anaconda navigator отвориће се прозор као на слици 5, у коме је могуће погледати окружење base(root) и у оквиру њега библиотеке које је Anaconda већ инсталирала за вас.



*Слика 5 :*

*Anaconda navigator преглед*

Пре него што отворимо радно окружење потребно је инсталирати неколико библиотека уколико оне по дифолту већ нису инсталиране (то можете погледати у инсталираним пакетима у Anaconda navigator-у, у base-root окружењу или виртуалном окружењу уколико сте га сами креирали)

У Windows старт менију укуцајте Anaconda и изаберите Anaconda prompt конзолу.

Укуцајте следеће команде:

> conda install numpy

> conda install scikit-learn (Python (>= 3.5) и NumPy (>= 1.11.0))

> conda update scikit-learn

> conda install matplotlib

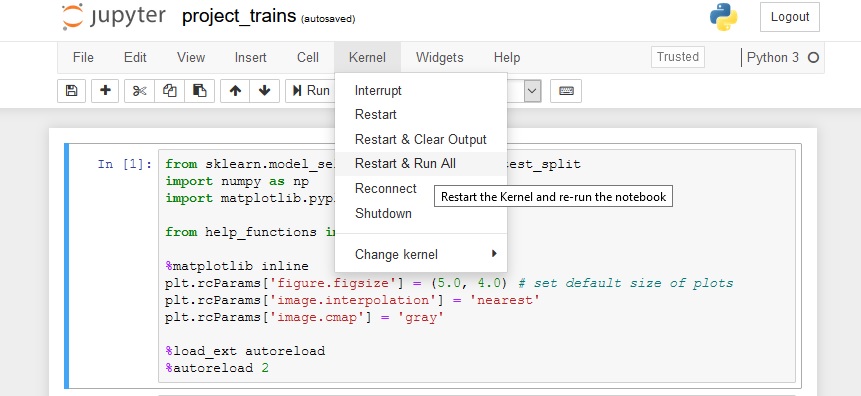
> conda install -c conda-forge jupyterlab (Python (>= 3.3) и Python 2.7)

(Библиотеке је могуће инсталирати и у оквиру Anaconda navigator-а)

Једна од библиотека, коришћена у овом пројекту, је jupyter notebook. Ово је open-source веб апликација која омогућава “live” код, једначине, визуализацију и наративан текст на једном месту.

Како би отворили јупитер свеску у конзоли укуцајте команду > jupyter notebook

Приметиће да се у дифолт веб прегледачу отворио локални порт и да вам је на рачунару омогућен локални сервер. Преусмерите радно окружење на директоријум где се налази пројекат који сте преузели и отворите јупитер свеску *“project\_trains.ipynb”.*

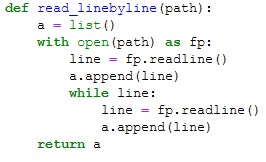
Сада је могуће извршити сваку од линија кликом на опцију Run (једна по једна линија) или Kernel->Restart and Run All (слика 6).

*Слика 6 :*

*Рестартовање кернела и покретање свеске*

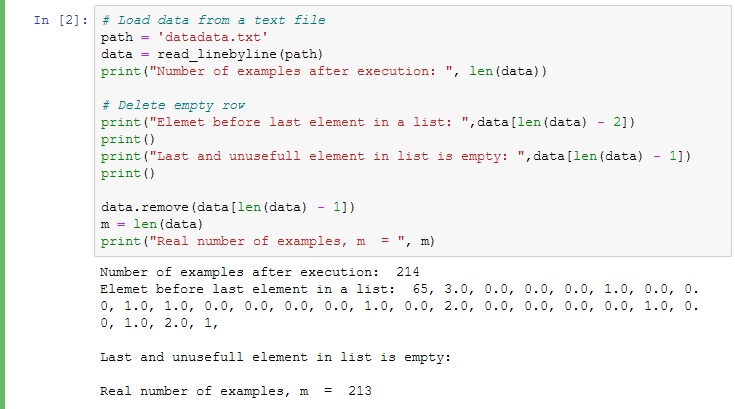
3. Обрада и припрема података

Учитавање података из текстуалног фајла *“datadata.txt”* (модом “line by line”) омогућава нам функција која је учитана у свеску из фајла *“help\_functions.py”* (слика 8).

У овом python фајлу она изгледа као на слици 7.

*Слика 7 :*

*Отварање фајла, читање фајла по моду “линија по линија” и складиштење редова у листу*

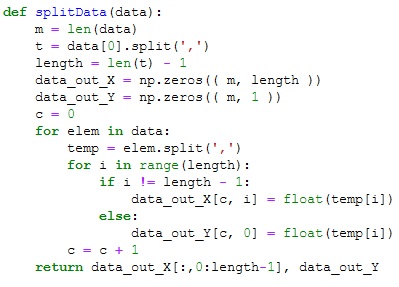


*Слика 8 :*

*Учитавање података, брисање последње празне линије и преузимање броја инстанци ‘m’*

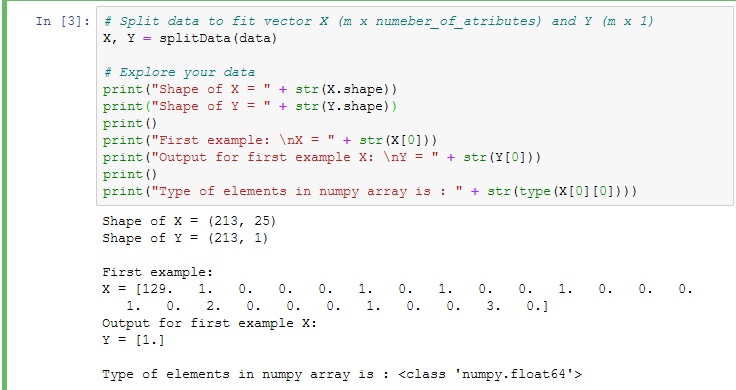
Следећи таск је раздвајање свих улазних и излазних X и Y података, како би добили два вектора димензија редом (213, 25) и (213, 1), где је вредност 213 аутоматски преузета вредност као број инстанци и вредност 25 аутоматски преузета вредност као број атрибута за сваку од инстанци.

Функција *splitData(data)* нам омогућава ово раздвајање (слика 9).



*Слика 9 :*

*Креирање два numpy вектора за сет свих података X и Y*

Преглед добијених података и њихов тип приказан је на слици 10:

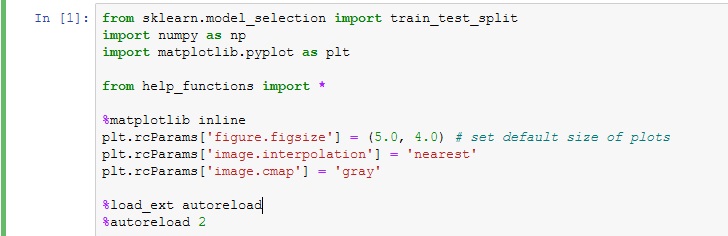
*Слика 10 :*

*Сет свих података X и Y преузетих из фајла “datadata.txt”*

За тренинг неуронске мреже из скупа X и Y изабрано је 80 посто података, док је за тестирање неуронске мреже изабрано преосталих 20 посто. Функција која нам је омогућила ово раздвајање налази се у пакету *sklearn.model\_selection* и она је на почетку свеске импортована помоћу наредбе

> from sklearn.model\_selection import train\_test\_split

као што се може видети на почетку свеске где се налазе наредбе и за остале пакете (слика 11).

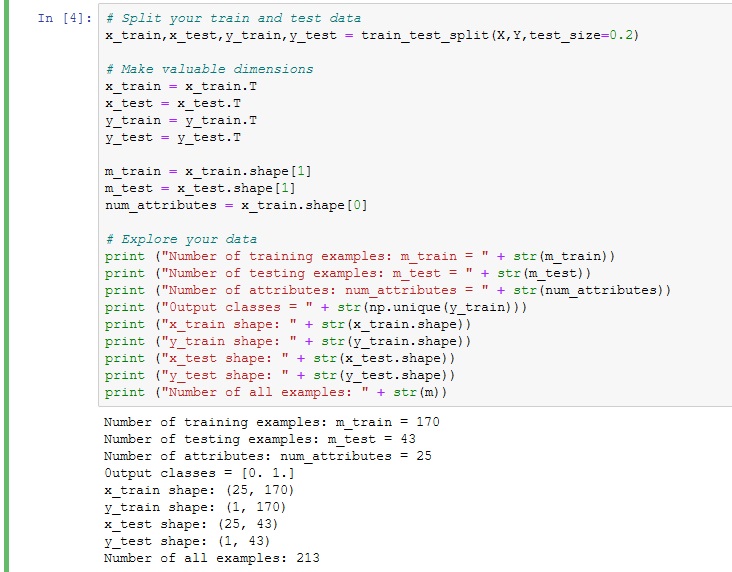


*Слика 11 :*

*Импортовање потребних библиотека и подешавање параметара за плотовање слика*

Након успешног извршавања функције *train\_test\_split(X,Y,test\_size=0.2)* подаци који служе за тренинг налазе се у 2D numpy векторима x\_train и y\_train, док су подаци за тест смештени у векторима x\_test и y\_test (слика 12).

Преглед броја тренинг/тест примера, броја атрибута и димензија поменутих numpy вектора налази се на слици 12.



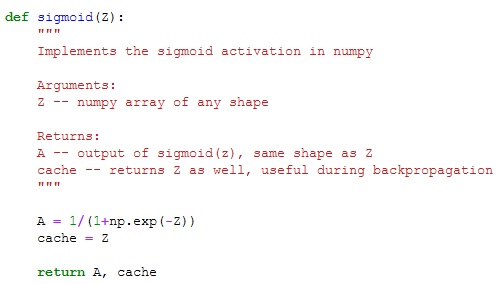
*Слика 12 :*

*Преглед података који ће се користити за тренинг и тест неуронске мреже*

4. Архитектура неуронске мреже

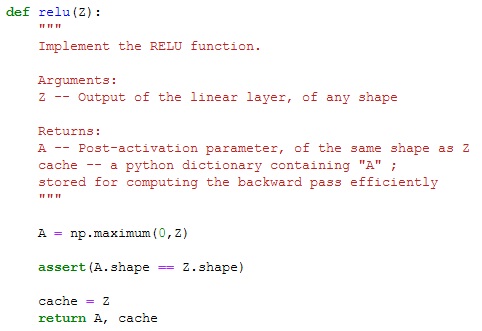
Модел неуронске мреже представљају 4 скривена слоја и један излазни слој.

Активациона функција која је коришћена у скривеним слојевима је ***relu*** нелинеарна функција, а на излазу ***sigmoid*** нелинеарна функција (слика 13 и 14).



*Слика 13 :*

*Имплементација sigmoid активационе функције*



*Слика 14 :*

*Имплементација relu активационе функције*

*Зашто су изабране баш ове активационе функције?*

4. 1. Активационе функције

У скривеним слојевима ***relu*** функција се боље приказала зато што има бољу перформансу конвергенције у односу на ***sigmoid***, ефективнија је приликом израчунавања јер она само треба да покупи максимум вредност између вредности 0 и x, где је x улазна променљива ове функције.

Главна предност функције ***relu*** је мања вероватноћа да дође до нестајања односно експлодирања градијента. Уколико би као активациону функцију изабрали линеарну функцију ***g(z) = z*** (што није пракса у скривеним слојевима) и вектор b изједначили са 0 вектором, онда би наше предвиђање y\_hat било:

|  |  |
| --- | --- |
| 0.5 | 0 |
| 0 | 0.5 |

**y\_hat = \* \* \* … \* \* \* \* x** , где означава нумеричку вредност за последњи слој , а \* x је заправо што је g(), итд.

Уколико сваку матрицу изједначимо са вектором ,

важиће следеће:

|  |  |
| --- | --- |
| **0.5** | **0** |
| **0** | **0.5** |

**y\_hat = \* на степен (L-1) \* x,**

што значи да ће наше предвиђање имати степен опадања .

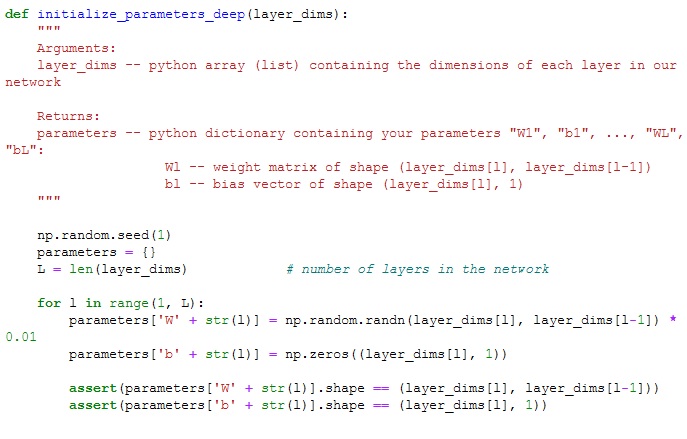
Уколико су параметри < (јединична матрица) за дубоке неуронске мреже активациона функција ће се експоненцијално смањивати, односно расти уколико уместо 0.5 имамо вредност 1.5, где је степен раста и > . У првом случају активационе функције неуронске мреже ће нестати, у другом случају експлодирати. Самим тим ће и градијенти бити мањи односно већи, што дубље улазимо у неуронску мрежу. Како ови параметри зависе од дубине мреже тј. броја слојева било би добро да на иницијализацију вектора (чија је димензија у слоју једнака броју излаза из тог слоја пута број улаза у тај слој) у слоју утиче управо број неурона у слоју . Стога, уколико варијанса, за relu активациону функцију, буде , где је број неурона у слоју, улази у активациону функцију биће грубо око нуле, што значи да ће и бити у том опсегу.

На овај начин смо редуковали нестајање/експлодирање градијента, али га нисмо у потпуности искоренили.

На слици 15 и 16 приказане су функције иницијализације параметара (параметре прослеђујемо као python-ов речник layers\_dims са кључевима :

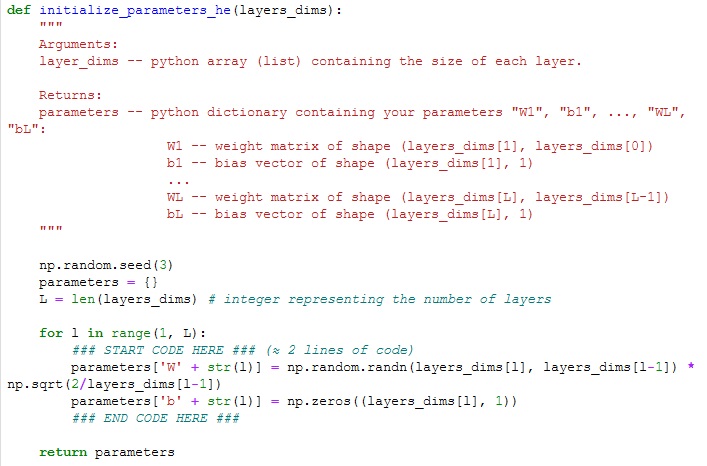
Прва слика приказује random иницијализацију параметара која не регулише експлодирање/нестајање градијената. Random вредности параметара помножене су затим са 0.01. Разлог овог додатног множења је уколико је параметар јако велики приликом рачунања вредности, вредност на графику за параметар је јако далеко од нуле, што успорава учење (градијент опадања). Овај додатни параметар је хипер-параметар, што значи да мора да се подешава, што је још један разлог мање да користимо ову иницијализацију насупрот ‘he’ (‘хе’ ћир.) иницијализацији која не садржи хипер-параметре.

Друга слика приказује ‘he’ иницијализацију која се показала најбољом за дубоке мреже. Иако наша мрежа није јако дубока (са само 4 скривена слоја), ипак се испоставља да је тачност мреже већа уколико користимо ‘he’ иницијализацију параметара (доказ касније – *глава 5*).



*Слика 15 :*

*Random иницијализација параметара*

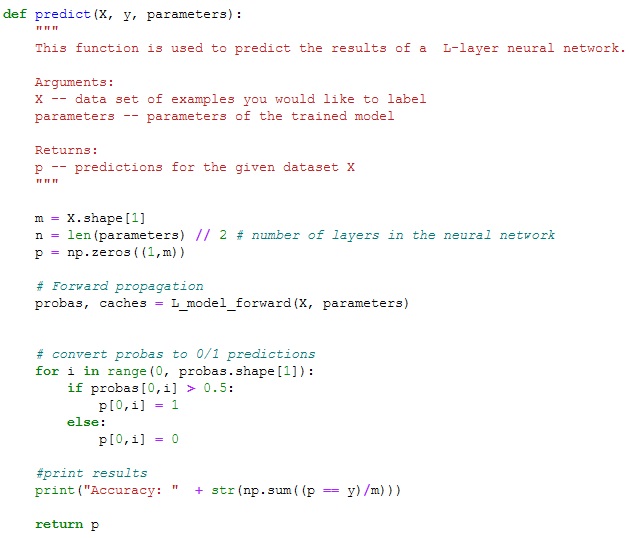


*Слика 16 :*

*‘He’ иницијализација параметара*

***Sigmoid*** функција изабрана је јер на излазу очекујемо вредности између 0 и 1, што је опсег излазних вредности ове функције. Уколико је на излазу вредност већа од 0.5 вредност излаза биће 1, обрнуто 0. На тај начин креирамо предвиђање за сваку од инстанци. Функција која нам омогућава предвиђање назива се *predict* и дефинише као *predict (X, y, parameters)*

Она изгледа као на слици 17.



*Слика 17 :*

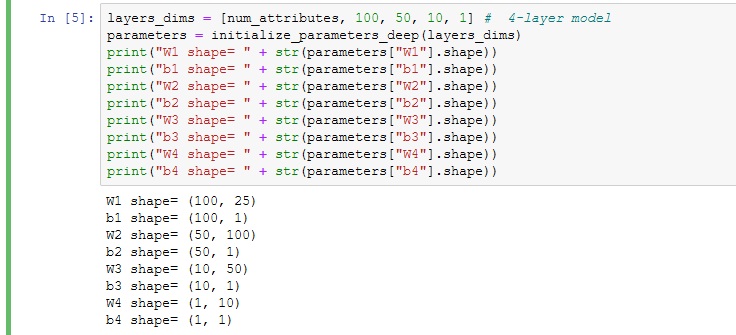
*Предвиђање на тренинг и тест скупу података и одређивање тачности мреже на ова два скупа*

4. 2. Параметри за тренирање

Као што је напоменуто, број скривених слојева у моделу неуронске мреже је 4.

Они су креирани од стране функције са слике 13 (за излазни слој са слике 14). Обе функције дефинишу исту димензију, али различите вредности параметара.

Зарад приказивања димензија свих параметара извршићемо функцију са слике 13 и приказати њихове димензије (слика 18).

**

*Слика 18:*

*Приказ димензија свих параметара у мрежи*

5. Неуронска мрежа са L слојева

*Сумирано :*

* Улазне вредности су вектори величина 25 x 170 за тренинг и 25 x 43 за тест
* Излазне вредности су вектори величина 1 x 170 за тренинг и 1 x 43 за тест
* Улазни вектор X је транспонован и помножен са вектором тежина , затим је овом производу додат вектор . Резултат се назива линеарна јединица .
* Резултат линеарне јединице прихвата ***relu*** активациона функција. Овај процес се понавља неколико пута за сваки пар (), у зависности од архитектуре модела
* На крају ***sigmoid*** активациона функција преузима последњу линеарну јединицу . Уколико је излаз ***sigmoid*** функције > 0.5 класифицира се као воз који иде на исток (обрнуто запад)

*Генерална методологија мреже :*

1. Иницијализација параметара / Дефинисање параметара
2. Петља кроз број итерација

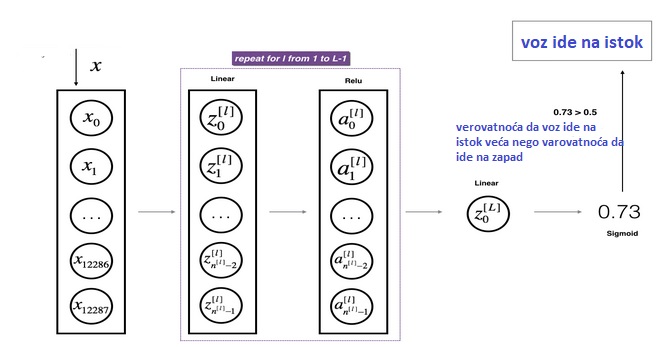
2.1 Пропагација унапред

2.2 Рачунање трошка параметара

2.3 Пропагација уназад

2.4 Update-овање параметара (користећи параметре и градијенте из пропагације уназад)

1. Штампање трошка параметара на сваких 100 итерација
2. Коришћење истренираних параметара за предвиђање

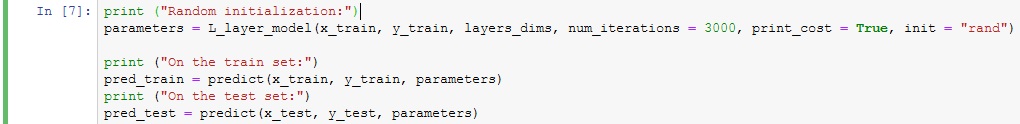


*Слика 19 :*

*Архитектура неуронске мреже са L слојева*

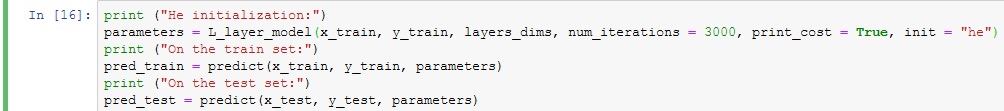
Приказ модела уколико се бира random, односно ‘he’ иницијализација може се погледати на слици 28.

Први, неунапређени модел неуронске мреже прихвата X и Y тренинг податке (x\_train, y\_train), речник који се састоји од фиксних димензија свих параметара који су употребљени у мрежи (layers\_dims), броја итерација (num\_iterations), параметра који одлучује о томе да ли ће се трошак параметара приказати на графику (print\_cost), параметара који одлучује о типу иницијализације параметара тј. да ли је избор random или je избор ‘he’ иницијализација (init) и параметра који одлучује да ли ће се укључити L2 регуларизација (слика 20 и слика 21).



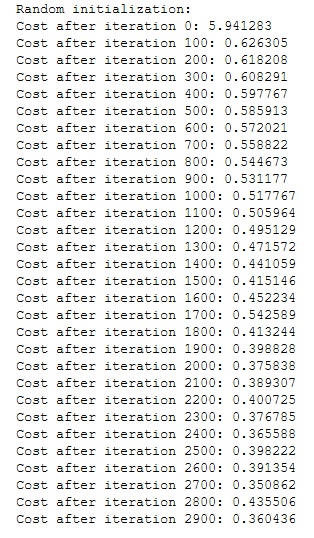
*Слика 20 :*

*Тренирање параметара када су они иницијализовани random методом*

*Слика 21 :*

*Тренирање параметара када су они иницијализовани помоћу ‘he’ иницијализације*

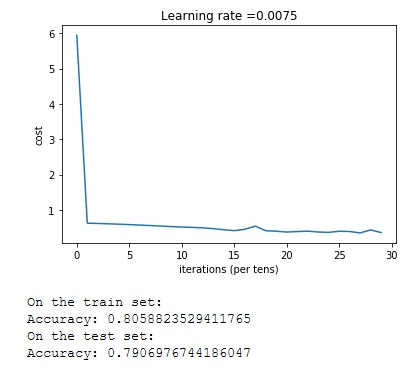
*Метода random иницијализације :*

Трошак параметара (cost) се смањује и после 3000 итерација достиже вредност око 0.35. Већ после 2000 итерација модел достиже своју највећу прецизност и трошак креће да “конвергира” око 0.35. Заправо трошак никад неће конвергирати у вредност минимума трошка јер за стопу учења изабрана је фиксна вредност. Због тога вредност трошка (cost) се стално “врти” око минимума и никада не достиже минимум (слика 22 и слика 23).

*Слика 22 :*

*Приказ трошка параметара након 3000 итерација када је изабрана random иницијализација*

Тачност овог модела на тренинг сету података је 0.805882, док је на тест сету података 0.79069 (слика 23).



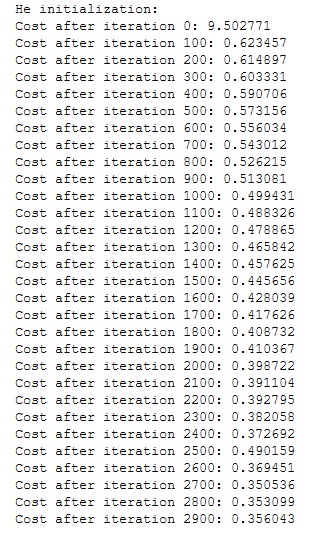
*Слика 23 :*

*Приказ функције трошка параметара након 3000 итерација са стопом учења 0.0075*

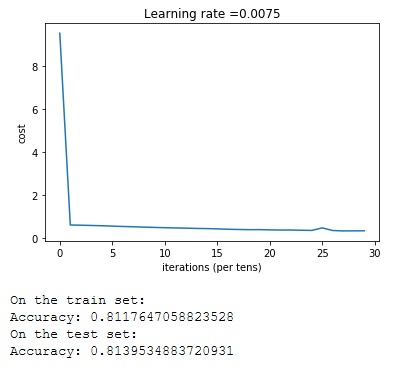
*Приказ тачности модела на тренинг и тест сету*

*‘He’ иницијализација:*

За исти број итерација, као и исту стопу учења истрениран је модел када су параметри иницијализовани помоћу ‘he’ методе иницијализације. Трошак параметара је приказан на слици 24, а график функције трошка као и прецизност модела када је изабрана ова метода на слици 25.

**

*Слика 24 :*

*Приказ трошка параметара након 3000 итерација када је изабрана ‘he’ иницијализација*

*Слика 25 :*

*Приказ функције трошка параметара након 3000 итерација са стопом учења 0.0075*

*Приказ тачности модела на тренинг и тест сету*

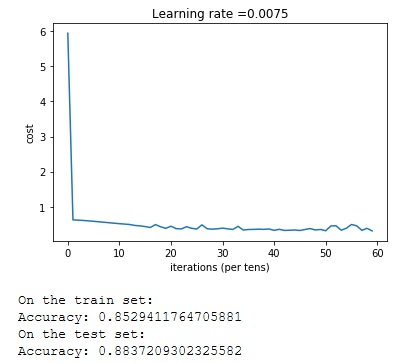
Може се видети да је тачност модела, када је изабрана ‘he’ иницијализација, на тренинг скупу 0.81176 већа од 0.80588 (random метода). Такође тачност модела на тест сету од 0.81395 је већа од тачности модела са random иницијализованим параметрима која је 0.79069.

Како је грешка на тест и тренинг скупу око 0.2 може се закључити да модел има високо одступање. Тренинг сет говори о одступању (bias, енгл.). Грешка услед високог одступања је количина колико се предвиђање очекиваног модела разликује од праве вредности тренинг података. Модел се може побољшати помоћу неколико опција:

1. Покушај са већом мрежом
2. Итерирај дуже

Уколико би број итерација уместо 3000 био 6000 први модел би дао тачност као на слици 26, а други модел као на слици 27

Такође параметар reg у модулу све време остаје None, јер није потребна регуларизација. Модел није склон “overfitting”-у, односно не трпи проблем високог варирања.





*Слика 26 : Слика 27 :*

*Поновљени тренинг са random Поновљени тренинг са ‘he’*

*иницијализацијом иницијализацијом*



*Слика 28 :*

*Функција модела са L слојева*

5. 1. Закључак

Може се закључити да иницијализација параметара утиче на понашање самог модела, на његову тачност при одређивању минималног трошка параметара. Различите иницијализације доводе до различитих резултата.

Random иницијализација је добра и користи се за разбијање симетрије и осигуравање да различите скривене јединици могу научити различите ствари, али се не користи уколико имамо параметре који су јако великих димензија.

‘He’ иницијализација са друге стране ради добро за мреже које за активационе функције имају ***relu*** нелинеарну функцију.

6. Оптимизована неуронска мрежа са L слојева

*Генерална методологија мреже :*

1. Иницијализација параметара / Дефинисање параметара
2. Иницијализација оптимизатора (ако је потребно)
3. Петља кроз број епоха

3.1 Избор мини серија улазних и излазних података (mini batches)

3.2 Петља за сваки пар серија података (minibatch\_X, minibatch\_Y)

3.3 Пропагација унапред

3.4 Рачунање трошка параметара (и регуларизација, ако је потребно)

3.5 Пропагација уназад

3.6 Избор оптимизатора (*градијент опадања, моментум* или *адам*)

3.6.1. Update-овање параметара (користећи параметре и градијенте из пропагације уназад)

4. Штампање трошка параметара на сваких 100 епоха

5. Коришћење истренираних параметара за предвиђање

6. 1. Иницијализација параметара / Дефинисање параметара

За иницијализацију параметра изабрана метода је ‘he’ иницијализација како се показала бољом методом за дубоке неуронске мреже у односу на random иницијализацију.

6. 3. Петља кроз број епоха

6. 3. 1. Избор мини серија улазних и излазних података

6. 3. 2. Петља за сваки пар серија података

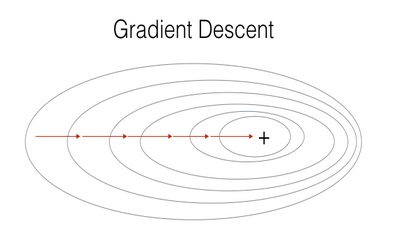
*Minibatch градијент опадања :*

Унапређени модел користи серије улазних података као парове (minibatch\_X, minibatch\_Y). Разлика у односу на класичан алгоритам градијента опадања је што градијент опадања даје резултат тј. вредност трошка параметара (cost) тек након што процесуира цео сет података. Уколико је m укупан број тренинг примера (цео сет података), а num\_iter број итерација, овај алгоритам ће након једне итерације (за num\_iter = 1), процесуирати цео сет података од m примера и направити мали корак оптимизацији трошка (трошак ће бити оптимизован само једном). Да би оптимизовао укупан трошак овај алгоритам ће процесуирати цео сет података онолико пута колики је дефинисани, максимални број итерација. То значи да градијент опадања процесуира цео сет тренинг података (“hole batch”) истовремено и понавља ово num\_iter пута.

Minibatch градијент опадања користи серије тренинг података. Овај алгоритам процесуира део сета тренинг података (minibacth) истовремено и прерачунава трошак онолико пута колико има minibacth-ова и понавља ово num\_iter пута.

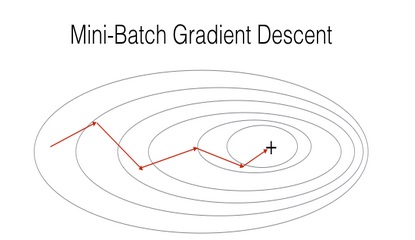
Функција трошка код класичног градијента опадања у времену стално опада. Уколико то није случај онда сигурно нешто није у реду.

Функција трошка код minibacth градијента опадања у времену не опада стално. Разлог томе је што приликом рачунања трошка сваки пут радимо са другачијим сетом података (minibacth-евима) и трошак се update-ује након процесуирања сваке серије. Због овог разлога добијамо функцију која у себи носи шум (слика 29 и 30).



*Слика 29 :*

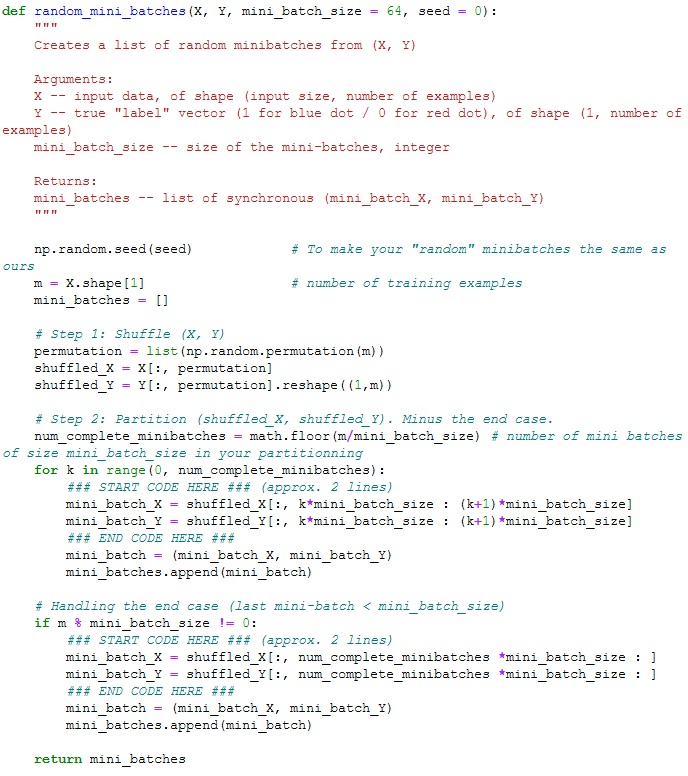
*Осцилација параметара не постоји. Функција трошка је опадајућа функција*



*Слика 30 :*

*Осцилација параметара постоји. Функција трошка је није константно опадајућа функција*

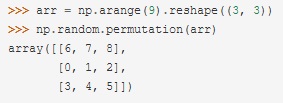
Како би поделили тренинг скуп тренинг подата на серије података креирана је функција *random\_mini\_batches(X, Y, mini\_batch\_size, seed)* (слика 31).



*Слика 31 :*

*Имплементација функције за креирање random серија тренинг података*

Функција са слике 31 креира листу permutation у којој се налазе random измешани бројеви од 0 до укупног броја инстанци што је у нашем случају 170. За пермутације користи се numpy функција као на слици 32, односно np.random.permutation(m)



*Слика 32 :*

*Приказ функције np.random.permutation(m)*

Затим за први ред вектора X (први атрибут за сваки од 170 тренинг примера) ова функција помеша 170 вредности у односу на листу пермутација. И тако за сваки ред вектора X. Векторизација у python-у омогућава избегавање петљи и мешање података у само једном реду. Ово се односи и на вектор Y, како парови података (X и Y) не би изгубили веродостојност (то омогућава листа пермутација која има фиксне вредности од 0 до 169). На слици 31 измешани подаци су shuffled\_X и shuffled\_Y. Затим се у односу на изабрану величину серије података (колико тренинг података садржи једна серија) креира број који описује колико таквих серија постоји. Уколико се испостави да укупна количина тренинг података (m) није дељива са величином једне серије (mini\_batch\_size = 64), онда ће све серије (сем последње) имати исти број података (64), а последња серија онолико података колико је потребно да би се допунио број до 170 тренинг података (слика 33).



*Слика 33 :*

*Серије података од по 64 тренинг примера (64 за X И 64 за Y) и*

*последња серија који има (m/64 + 1) податак где је m укупан број тренинг примера*

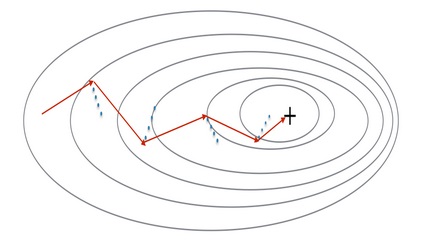
6. 2. Иницијализација оптимизатора

Није потребна иницијализација за алгоритам градијента опадања

*Момент*

Још један алгоритам који ради брже од minibatch градијента опадања је Момент, а настао је спајањем minibatch градијента опадања и алгоритма за експоненцијално отежано усредњавање. Идеја овог алгоритма је да редукује осцилације које minibatch градијент опадања прави, тј. да вредности разлика хоризонталних корака учини да буду близу 0 (разлике по вертикалној оси и даље ће бити велике).

На следећој слици црвене стрелице показују смер који прави један корак minibatch градијента опадања са Моментом. Плаве тачке приказују смер градијента при сваком кораку. Уместо да само прати градијент, пустићемо да на градијент утиче параметар *𝑣*  и онда направити корак у смеру *𝑣* (слика 34).



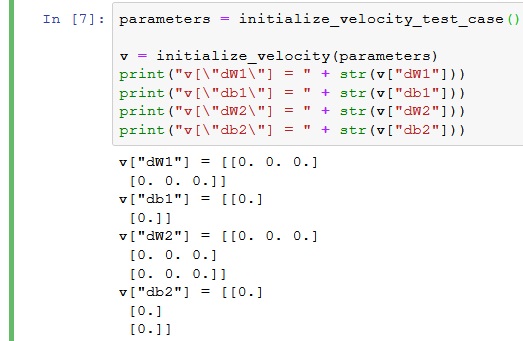
*Слика 34 :*

*Осцилација параметара. Функција трошка је под утицајем параметра 𝑣*

Користећи Момент алгоритам добијамо мање осцилације по вертикалној оси, али ће понашање корака бити брже по хоризонталној оси. Параметар *𝑣*  можемо да сматрамо као “брзину” лопте која се котрља низбрдо и развија брзину (и нагиб - момент) према смеру градијента тј. “нагиба брда”.

Иницијализација овог параметра је нула-иницијализација. Параметар је у коду представљен као python речник који је иницијализован са нула-низовима.

Ради примера показана је иницијализација овог параметра на слици 35.



*Слика 35 :*

*Иницијализација параметра 𝑣*

*Адам (Adaptive moment estimation)*

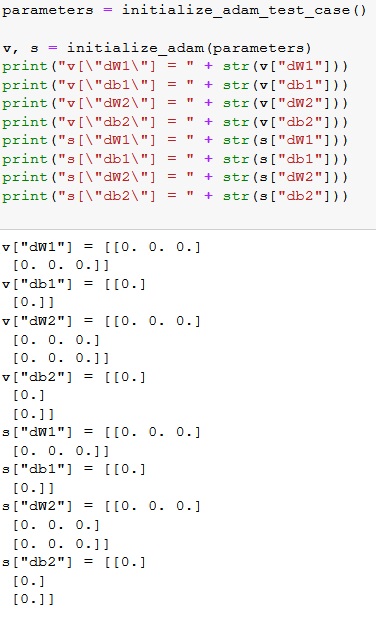
Овај алгоритам оптимизације настао је спајањем RMSprop (Root mean square prop) алгоритма и Момент алгоритма. RMSprop алгоритам чини оно што алгоритам Момент није успео да редукује, а то је да разлике/осцилације у вертикалним осама учини мањим. Када ова два алгоритма заједно раде (Адам) чине да учење буде ефективно за разне неуронске мреже и архитектуре неуронских мрежа.

Како ради?

1. Рачуна експоненцијално отежане просеке претходних градијената и складишти их у променљиву *𝑣*  (складиштење пре корекције која користи одступање ), и израчунава *𝑣𝑐𝑜𝑟𝑟𝑒𝑐𝑡𝑒𝑑*  (са корекцијом која користи одступање)
2. Рачуна експоненцијално отежане просеке квадрата претходних градијената и складишти их у променљиву *s*  (складиштење пре корекције са одступањем ), и израчунава *s𝑐𝑜𝑟𝑟𝑒𝑐𝑡𝑒𝑑*  (са корекцијом која користи одступање )
3. Update-ује параметре у правцу заснованом на комбиновању информација из 1. и 2.

Иницијализација Адам променљивих *𝑣*  и *s*  је нула-иницијализација, где сваки од ова два параметра представља python речник који је иницијализован са нула-низовима.

Ради примера показана је иницијализација ова два параметра на слици 36.



*Слика 36 :*

*Иницијализација параметра 𝑣 и s*

6. 3. 3. Пропагација унапред

Пропагација унапред нам омогућава да добијемо излаз неуронске мреже. Како функционише пропагација унапред заправо тако функционише и неуронска мрежа када треба да креира предвиђање за одређени улазни податак. Пропагација унапред имплементирана је сликама 37, 38 и 39.

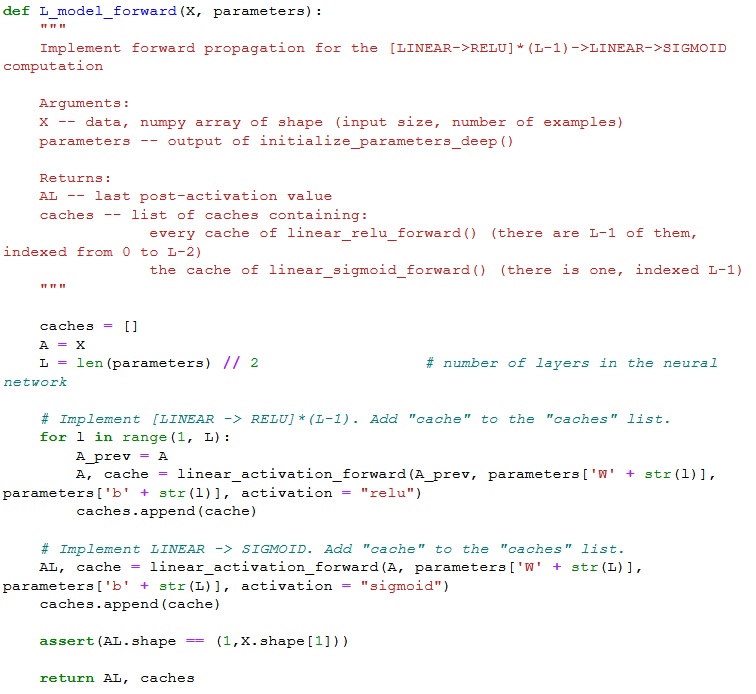
Активационе функције које се користе у скривеним слојевима у области неуронских мрежа не треба да буду линеарне.

*“Композиција сваке две линеарне функције је линеарна функција”*

Неуронска мрежа ће на тај начин израчунати линеарну активациону функцију улаза и понашаће се као да нема скривене слојеве.

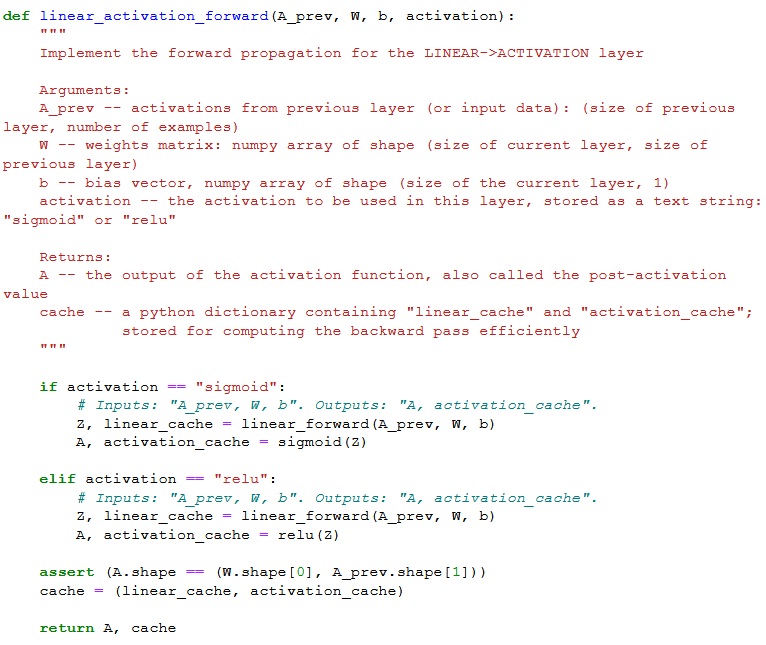
За границу одлучивања ово значи да се неће добро прилагодити подацима јер ће имати облик линије, а улазни подаци представљени на графику, чије се осе појединачно односе на сваки од улазних параметара за једну инстанцу, могу бити комплекснији и граница одлучивања која треба бити оптимизована градијентом опадања може бити јако комплексна.

Стога за активационе функције изабране су нелинеарне активационе функције ***relu*** и ***sigmoid*** јер се оне врло добро могу прилагодити било ком сету података.



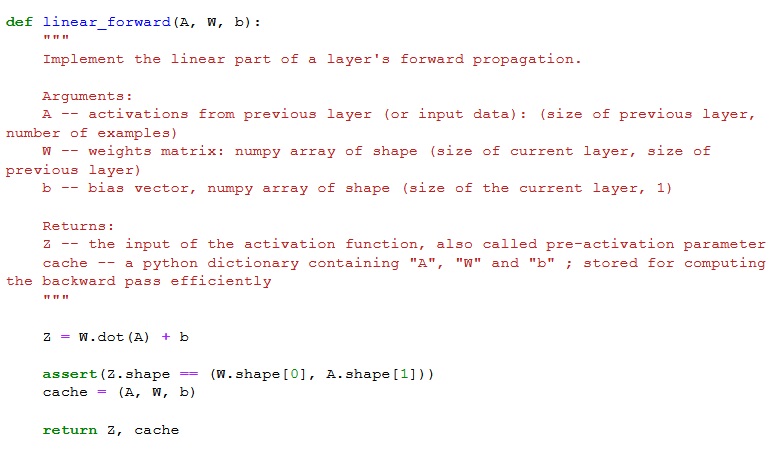
*Слика 37 :*

*Функција која омогућава пролаз кроз свих L слојева мреже*



*Слика 38 :*

*Функција која омогућава избор активационе функције у скривеним слојевима и рачунање вредности активационе функције у скривеним слојевима и на излазу*



*Слика 39 :*

*Функција која израчунава вредност линеарне функције пре активације*

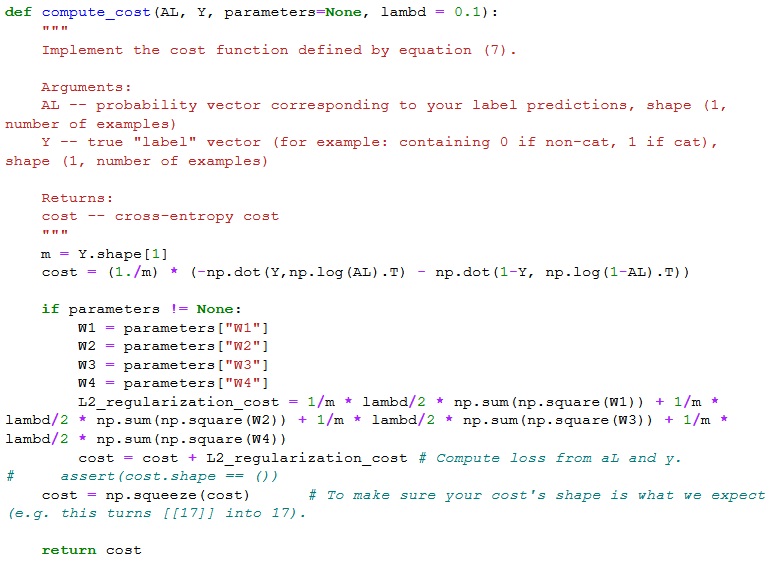
6. 3. 4. Рачунање трошка параметара

Логистичка регресија, за алгоритам градијента опадања (уопштено), користи другачију функцију трошка параметара него ону која се користи за линеарну регресију. Разлог томе је што квадратна функција грешке, којом се израчунава трошак параметара линеарне регресије, није конвексна за логистичку регресију.

Покретањем алгоритма градијента опадања са линеарном функцијом трошка довешће нас до погрешних резултата јер ће трошак параметара конвергирати у локалном минимуму, а не у оптималном минимуму који очекујемо.

Стога као функцију којом се израчунава трошак логистичке регресије изабрана је функција као на слици 40, са додатком који омогућава регуларизацију.

Регуларизација се примењује само на параметре из разлога што су параметри само обични бројеви, а параметри су високо димензиони параметри који пате од проблема “overfittinga” (слика 40). Уколико се утиче на параметре редуковањем њиховог утицаја помоћу регуларизације тиме се редукује утицај скривених неурона. На овај начин регуларизација спречава проблем “overfitting”-а јер спречава границу одлучивања да се уклапа у тешке/комплексне границе одлучивања.



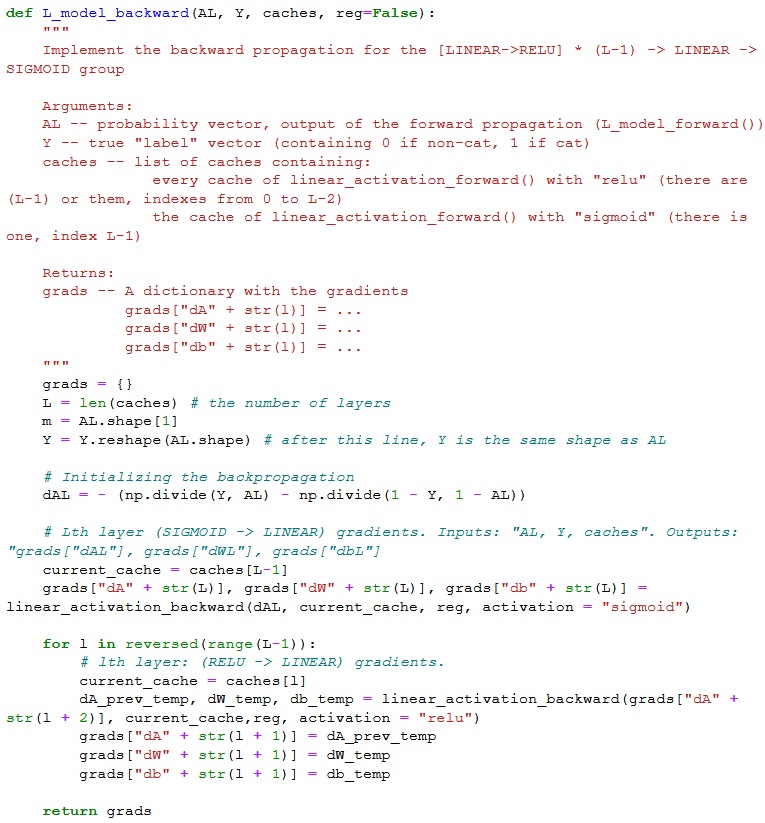
*Слика 40 :*

*Функција трошка параметара (cost функција) логистичке регресије*

6. 3. 5. Пропагација уназад

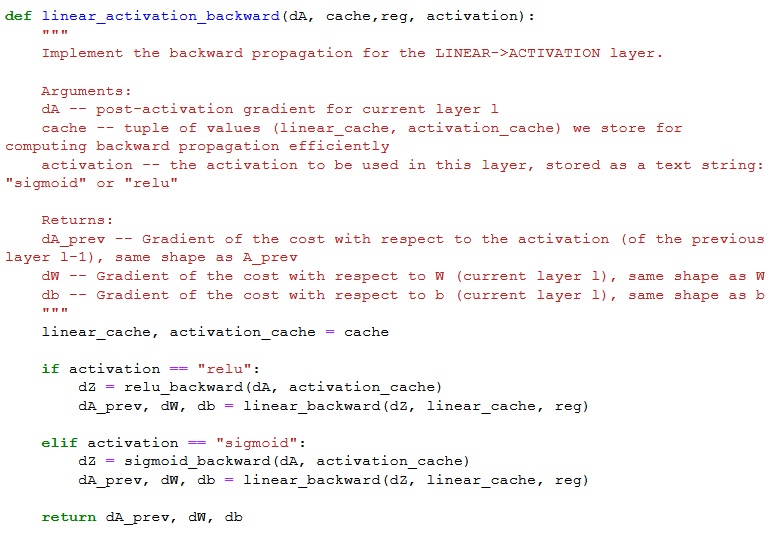
Пропагација уназад се користи да би се израчунали градијенти/изводи свих параметара у мрежи. Градијенти параметара су потребни како би се извршио update свих параметара у току оптимизације трошка параметара када се укључи један од алгоритма за оптимизацију.

Пропагација уназад је имплементирана на сликама 41, 42, 43 и 44



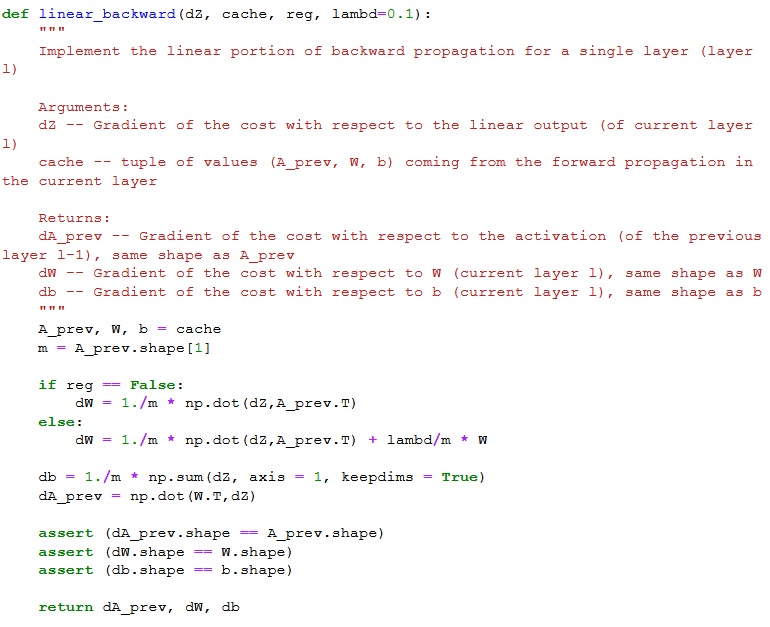
*Слика 41 :*

*Функција која израчунава градијент активационе функције и параметара на излазу и омогућава пролаз кроз свих L-1 слојева мреже*



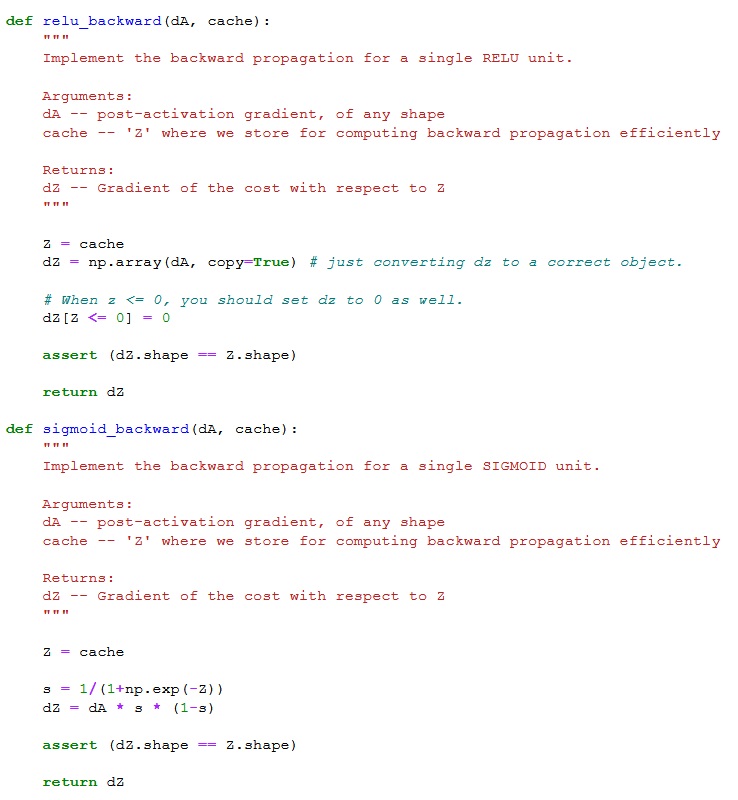
*Слика 42 :*

*Функција која омогућава подешавање активационе функције у скривеним слојевима и рачунање градијената активационих функција*



*Слика 43 :*

*Функција која израчунава градијент активационе функције слоја l-1 помоћу линеарних компутација*



*Слика 44 :*

*Имплементација пропагације уназад за један неурон и функције relu и sigmoid*

6. 3. 6 Избор оптимизационог алгоритма

6. 3. 6. 1. Update-овање параметара

Update-овање параметара се врши како би се пронашле оптималне вредности за све параметре који дефиниши границу одлучивања. Вредности најоптималнијих параметара треба да минимизују трошак параметара (cost).

*Градијент опадања*



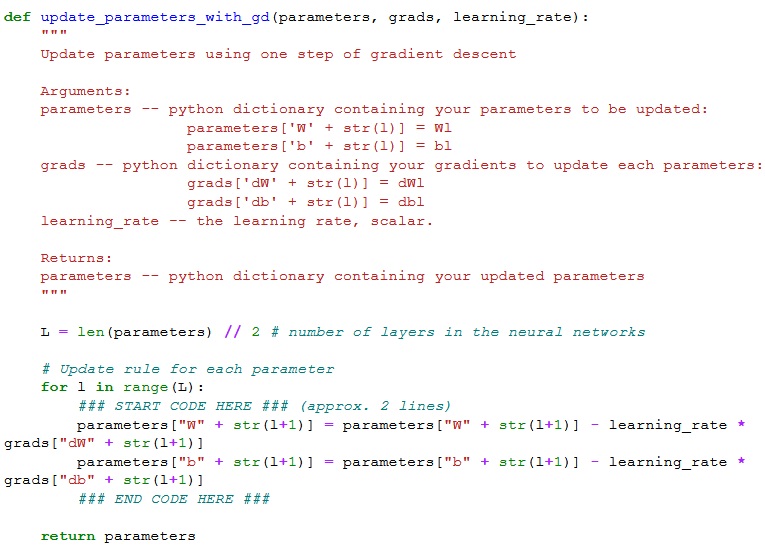
Правило градијента опадања : , где је 𝛼 стопа учења.

Како би update-овао параметре градијент опадања користи градијенте параметара који су претходно израчунати (слика 45).

Уколико је градијент параметра позитиван број следи да ће вредност претходног параметра бити већа од вредности update-ованог параметра па ће смер градијента опадања бити са десно у смеру доле-лево ка минимуму

Уколико је градијент параметра негативан број следи да ће вредност претходног параметра бити мања од вредности update-ованог параметра па ће смер градијента опадања бити са лево у смеру доле-десно ка минимуму.

Стога за било који знак градијента параметара градијент опадања ће тежити да минимизује параметре .

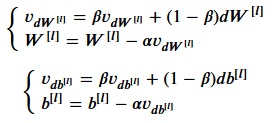
**

.

*Слика 45 :*

*Update-овање параметара користећи алгоритам градијента опадања*

*Момент*



Правило момента за = 1,…, : , где је

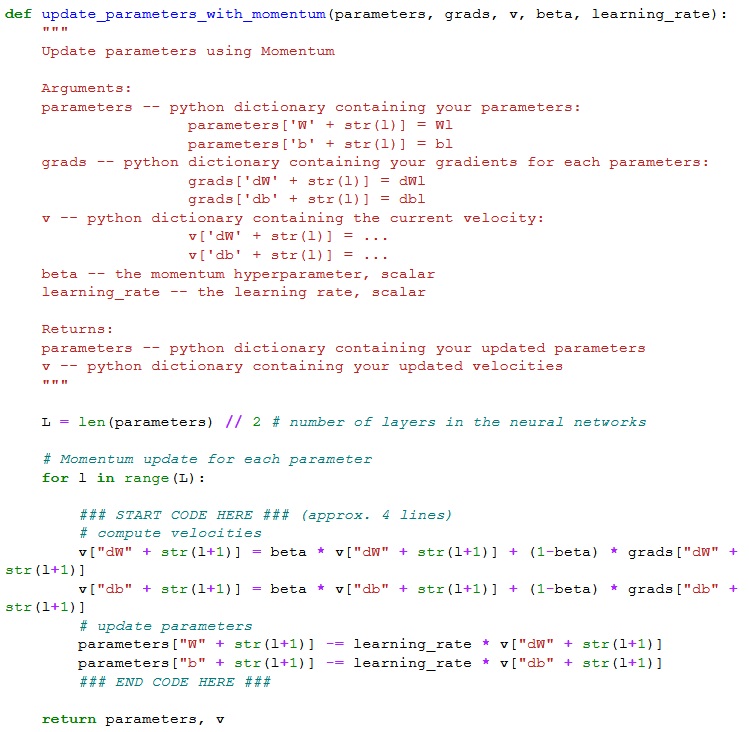
број слојева у мрежи,

𝛽 момент (“нагиб брда”),

𝛼 стопа учења и

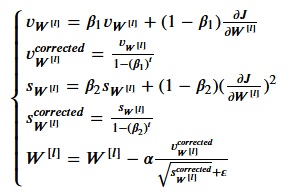
𝑣 параметар који утиче на смер градијента (“брзина лопте”)

Како би update-овао параметре алгоритам Момент користи градијенте параметара који су претходно израчунати (слика 46).



Слика 46 :

Update-овање параметара користећи алгоритам Момент



*Адам*

Правило Адама за = 1,…, , где је

броји број корака Адам

алгоритма,

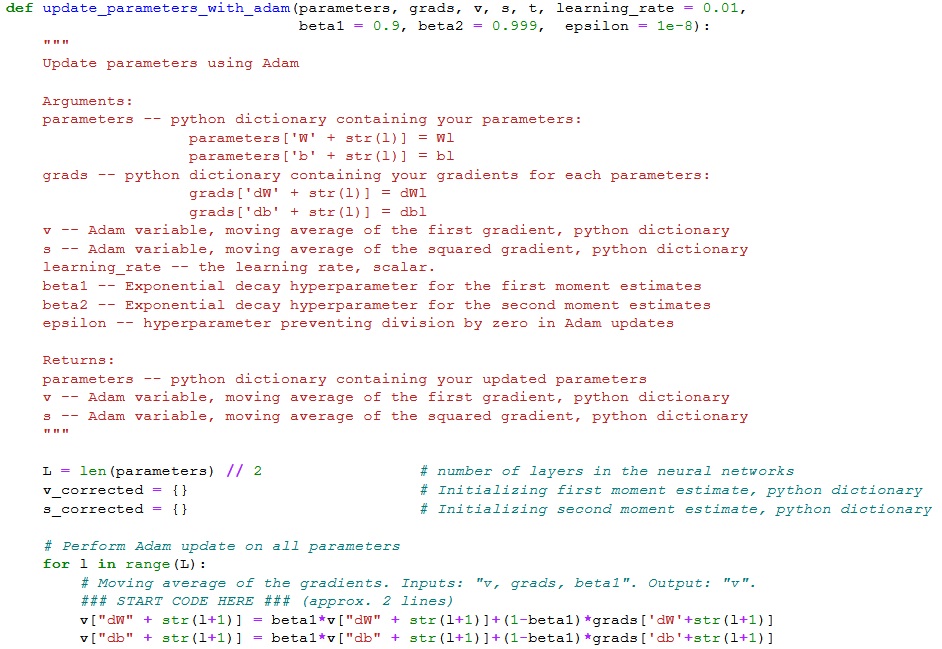
број слојева,

*𝛽*1 и *𝛽*2 су хипер параметри који контролишу експоненцијално отежане просеке,

стопа учења и

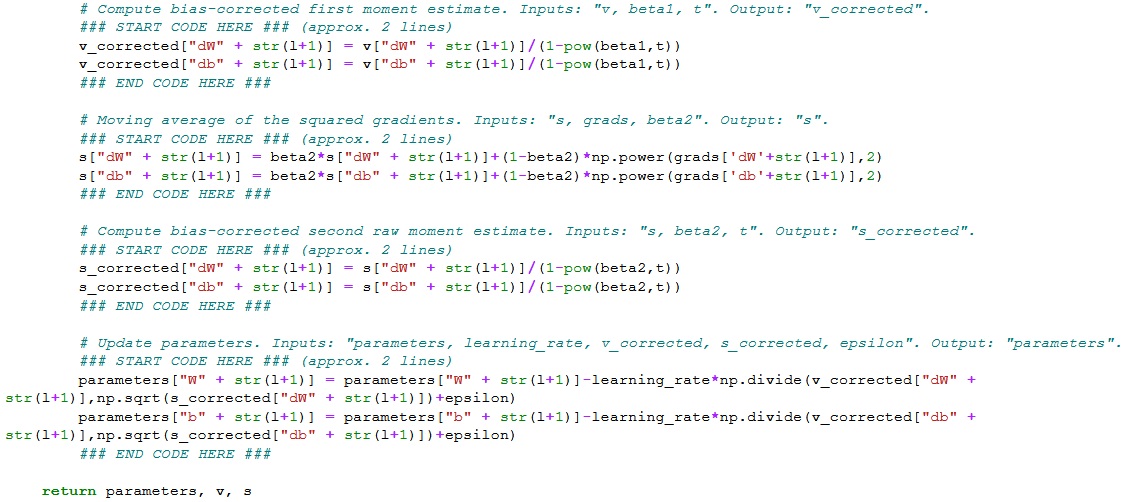
*𝜀* јако мали број да би се избегло дељење нулом

Како би update-овао параметре алгоритам Адам користи градијенте параметара који су претходно израчунати (слика 47 и 48).



*Слика 47 :*

*Update-овање параметара користећи алгоритам Адам, I део*



*Слика 48 :*

*Update-овање параметара користећи алгоритам Адам, II део*

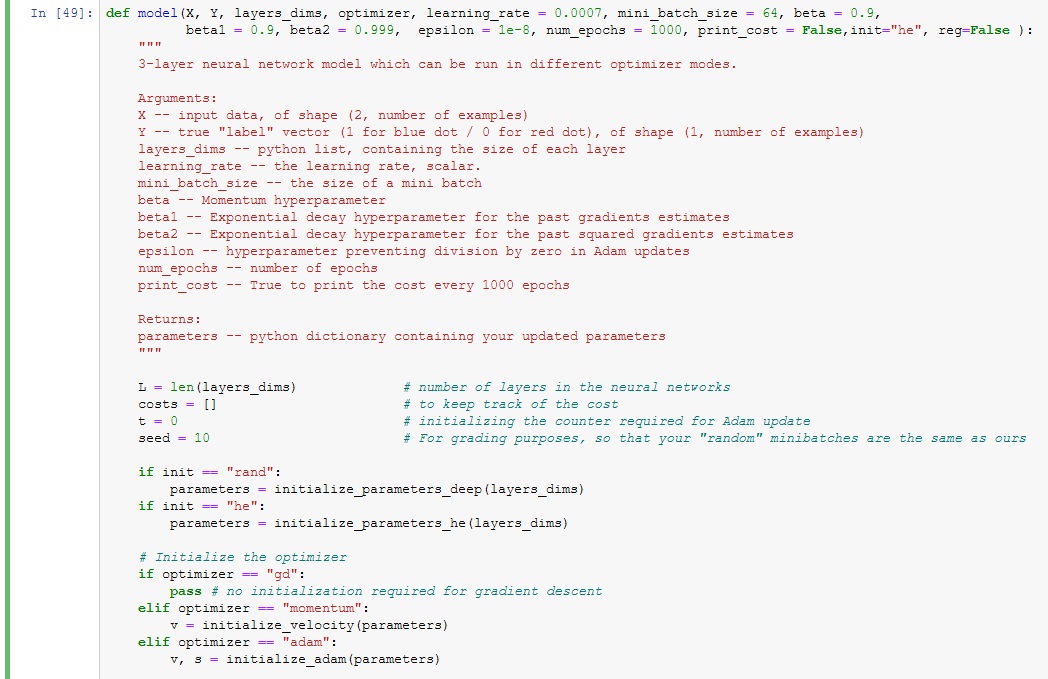
7. Модел неуронске мреже са L слојева

Све функције потребне за креирање модела неуронске мреже са L слојева налазе се у фајлу *“help\_functions.py”* и претходно су објашњене. Како би модел био истрениран све функције су позване у оквиру функције :

model(X, Y, layers\_dims, optimizer, learning\_rate = 0.0007, mini\_batch\_size = 64, beta = 0.9, beta1 = 0.9, beta2 = 0.999, epsilon = 1e-8, num\_epochs = 1000, print\_cost = False,init="he", reg=False ), где је

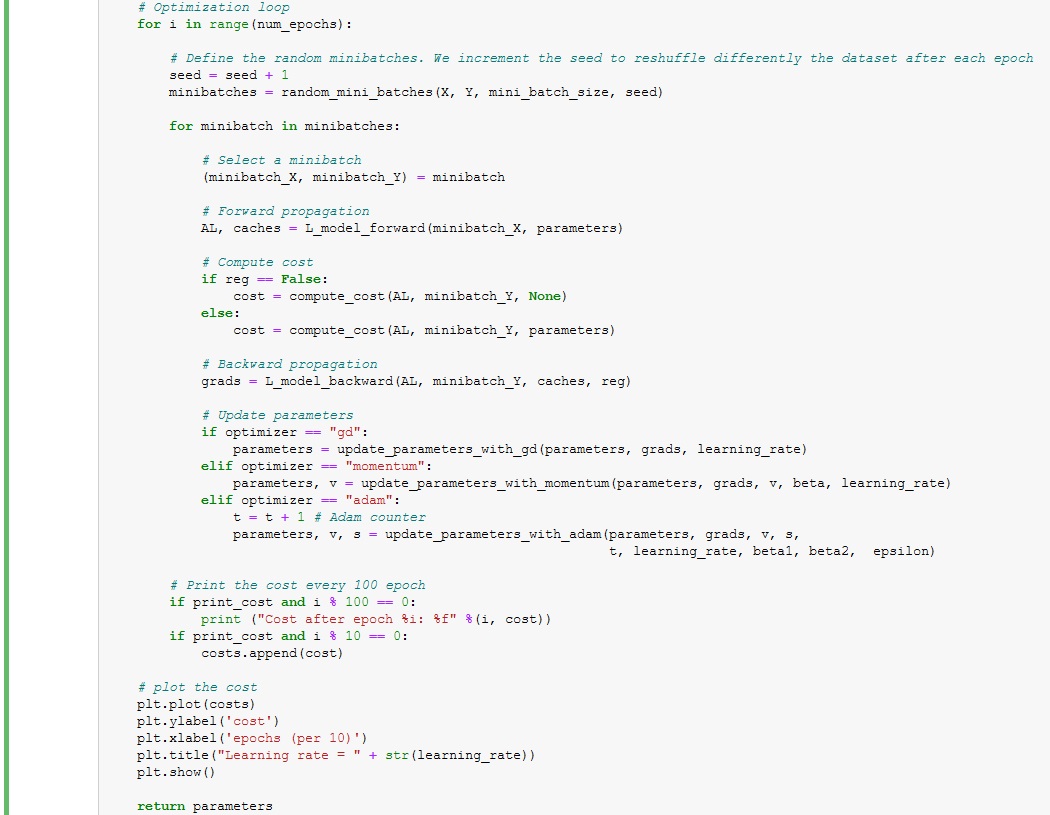
* X вредност свих улазних тренинг података
* Y вредност свих излазних тренинг података
* layers\_dims python-ов речник који садржи димензије параметара у скривеним слојевима
* optimizer параметар који одлучује о избору оптимизационог алгоритма
* learning\_rate = 0.0007 дифолт вредност стопе учења, уколико није наведена када се позове функција model (…)
* mini\_batch\_size = 64 дифолт вредност величине серије података, тј. колико података садржи једна серија
* beta = 0.9 дифолт вредност за хипер параметар алгоритма Момент
* beta1 = 0.9, beta2 = 0.999, epsilon = 1e-8 дифолт вредности за хипер параметре алгоритма Адам
* num\_epochs = 1000 дифолт вредност за број епоха
* print\_cost = False дифолт вредност за параметар који одлучује о цртању функције трошка
* init="he" дифолт вредност за параметар који одлучује о избору методе за иницијализацију параметара у мрежи
* reg=False дифолт вредност за параметар који може укључити регуларизацију параметра (као и , аутоматски)

Целокупан модел могуће је погледати на слици 49 и 50



*Слика 49 :*

*Модел оптимизоване неуронске мреже, I део*



*Слика 50 :*

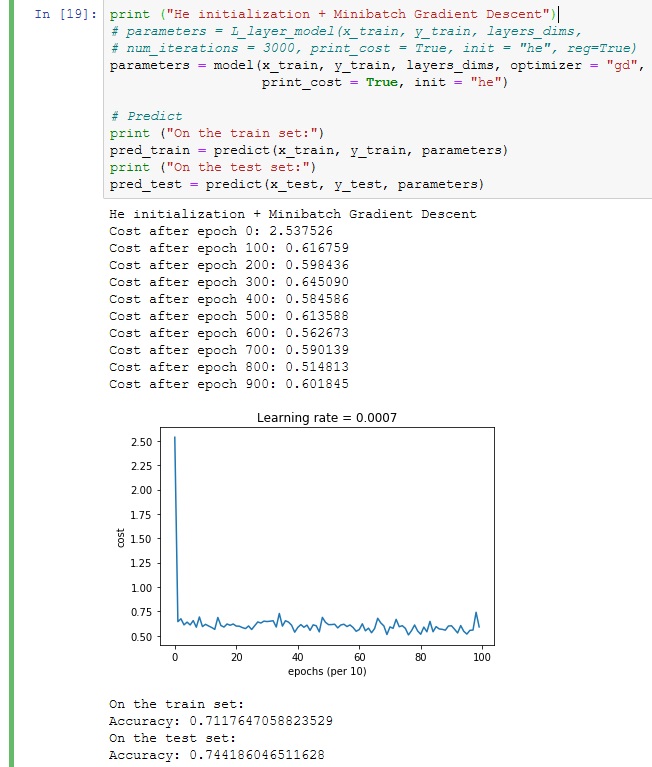
*Модел оптимизоване неуронске мреже, II део*

8. Тренирање и резултати неуронске мреже

Након 1000 епоха

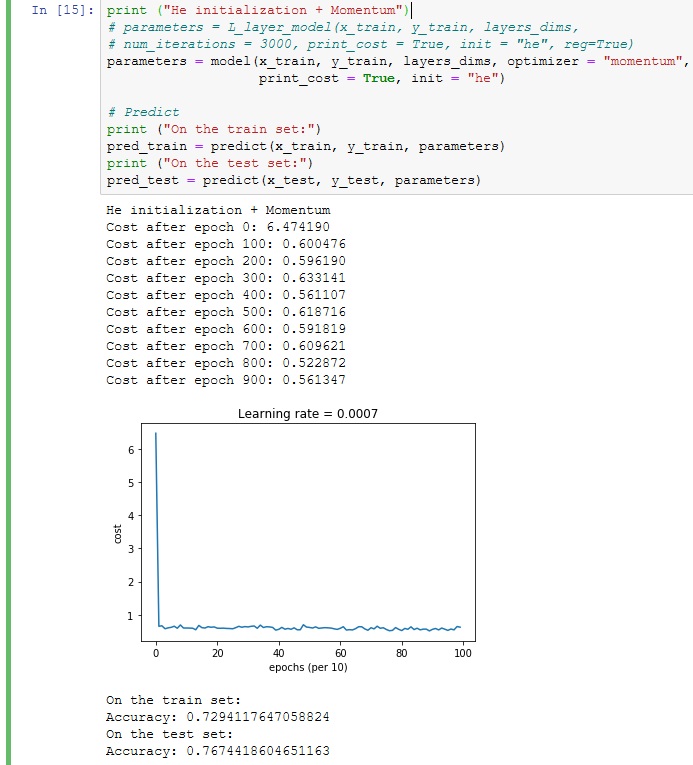
* алгоритам Minibatch градијент опадања има тачност на тренинг сету 0.7117647058823529, док је његова тачност на тест сету 0.744186046511628
* алгоритам Момент има тачност на тренинг сету 0.7294117647058824, док је његова тачност на тест сету 0.7674418604651163
* алгоритам Адам има тачност на тренинг сету 0.9529411764705882, док је његова тачност на тест сету 0.813953488372093

Резултати су дати на следећим сликама :



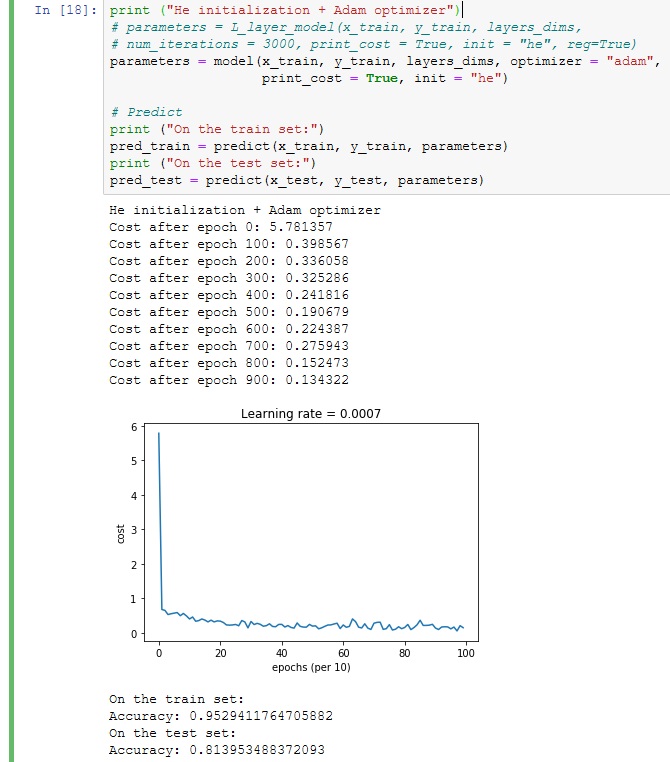
*Слика 51 :*

*Приказ функције трошка параметара и тачност алгоритма Minibatch градијента опадања на тренинг и тест сету*



*Слика 52 :*

*Приказ функције трошка параметара и тачност алгоритма Момент на тренинг и тест сету*



*Слика 53 :*

*Приказ функције трошка параметара и тачност алгоритма Адам на тренинг и тест сету*

9. Закључак

Овај рад је омогућио поглед на то како различити оптимизациони алгоритми утичу на прецизност неуронске мреже, понашање модела под различитим условима, који се тичу иницијализације свих параметара у мрежи, и доказе њеног понашања.

Рад је, заједно са свим изведеним доказима, омогућио закључак да је најбољи оптимизациони алгоритам Адам алгоритам за кога се испоставља да је најпрецизнији.

Иако неуронска мрежа не прелази “human-level” перформансу, и даље је њена прецизност велика, па се може закључити да ради врло добро под условима под којима је истренирана.

9. Литература

Преузето: Август 23, 2019, oд <https://www.coursera.org/search?query=deeplearning.ai>

Преузето: Август 25, 2019, oд <https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.math.html>

Преузето: Септембар 10, 2019, oд <https://www.geeksforgeeks.org/vectorization-in-python/>

Преузето: Септембар 10, 2019, од <https://arxiv.org/abs/1412.6980>